Model symulacji — raport częściowy — Projekt MOPS

Błażej Sewera, Mateusz Winnicki, Wojciech Kowalski

1 grudnia 2020

Projekt MOPS

Cele projektu

Celem projektu jest zasymulowanie węzła lub systemu dwóch węzłów sieciowych, obsługujących n źródeł ruchu typu ON-OFF, zdolnych generować pakiety o stałej długości L w stanie ON.

Każdy węzeł sieciowy modelujemy jako nieskończoną kolejkę FIFO, każdy ma podłączoną dowolną liczbę źródeł ruchu.

Wartości teoretyczne

Liczba pakietów w stanie ON jest opisywana rozkładem geometrycznym o funkcji rozkładu prawdopodobieństwa:

$$n_{ON} = (1 - p_{ON})^{k-1} p_{ON}$$

$$n_{ONavg} = \frac{1}{p_{ON}}$$

gdzie n_{ON} to liczba pakietów przesłana w jednej iteracji stanu ON. Odstęp między pakietami (czas interwału) jest stały i wynosi t_{int}.

Wtedy średni czas trwania stanu ON:

$$t_{ONavg} = \frac{t_{int}}{p_{ON}} + t_{prog}$$

gdzie t_{prog} to dodatkowy czas wprowadzany przez ograniczenia narzędzi programistycznych, implementacji i sprzętu, który dla uproszczenia pominiemy.

Średnia liczba bitów wysłanych przez źródło podczas jednego stanu ON wynosi:

$$n_{b\,avg} = \frac{L}{p_{ON}}$$

Długość stanu OFF opisana jest rozkładem wykładniczym.

$$t_{OFF} = \lambda e^{-\lambda t}$$

$$t_{OFFavg} = \frac{1}{\lambda}$$

W tym projekcie, dla uproszczenia posłużymy się analogicznym rozkładem geometrycznym dla długości stanu OFF, tylko w odróżnieniu od stanu ON, nie symulujemy wysyłania pakietów, a jedynie uwzględniamy interwał pomiędzy kolejnymi iteracjami tego stanu, czyli de facto bezczynnością.

$$t_{OFFavg} = \frac{1}{p_{OFF}} \cdot t_{int}$$

Z racji że stany ON i OFF następują naprzemiennie, średni czas trwania tych dwóch stanów:

$$t_{ON\,OFF\,avg} = \frac{t_{int}}{p_{ON}} + \frac{t_{int}}{p_{OFF}} = \frac{t_{int} \cdot (p_{ON} + p_{OFF})}{p_{ON} \cdot p_{OFF}}$$

Na podstawie powyższych danych można obliczyć średnią przepływność generowaną przez jedno źródło:

$$BR_{avg} = \frac{n_b}{t_{ON\,OFF\,avg}} = \frac{L p_{ON} p_{OFF}}{t_{int} p_{ON} (p_{ON} + p_{OFF})}$$

Zatem średnia szybkość napływu pakietów:

$$\frac{1}{\lambda} = \frac{BR_{avg}}{L} = \frac{p_{ON} p_{OFF}}{t_{int} p_{ON} (p_{ON} + p_{OFF})}$$

Więc przewidywany czas pomiędzy pakietami to:

$$\lambda = \frac{t_{int} p_{ON} (p_{ON} + p_{OFF})}{p_{ON} p_{OFF}}$$

Z racji że czas obsługi jednego pakietu jest stały ze względu na jego stałą szerokość, średni czas obsługi jest taki sam i wynosi μ.

Średnie obciążenie systemu:

$$\rho = \frac{\lambda}{\mu} = \frac{t_{int} p_{ON} (p_{ON} + p_{OFF})}{p_{ON} p_{OFF} \mu}$$

Prawdopodobieństwo, że w kolejce jest n pakietów, ma rozkład geometryczny:

$$P_n = (1 - \rho)\rho^n = \mathrm{Geo}(1 - \rho)$$

Badane metryki pomiarowe

- średnia liczba pakietów w kolejce l_{queue}
- średni czas oczekiwania w kolejce t_{wait}
- średnie opóźnienie przekazu pakietu, definiowane jako średnia suma czasu oczekiwania w kolejce oraz czasu odebrania całego pakietu przez węzeł sieciowy. Jako że długość pakietu jest stała i wynosi L, czas odebrania pakietu przez węzeł również będzie stały $t_{delay} = t_{wait} + T_{receive}$
- średnie obciążenie serwera

Przedstawiony model zaimplementujemy w języku programowania Python.

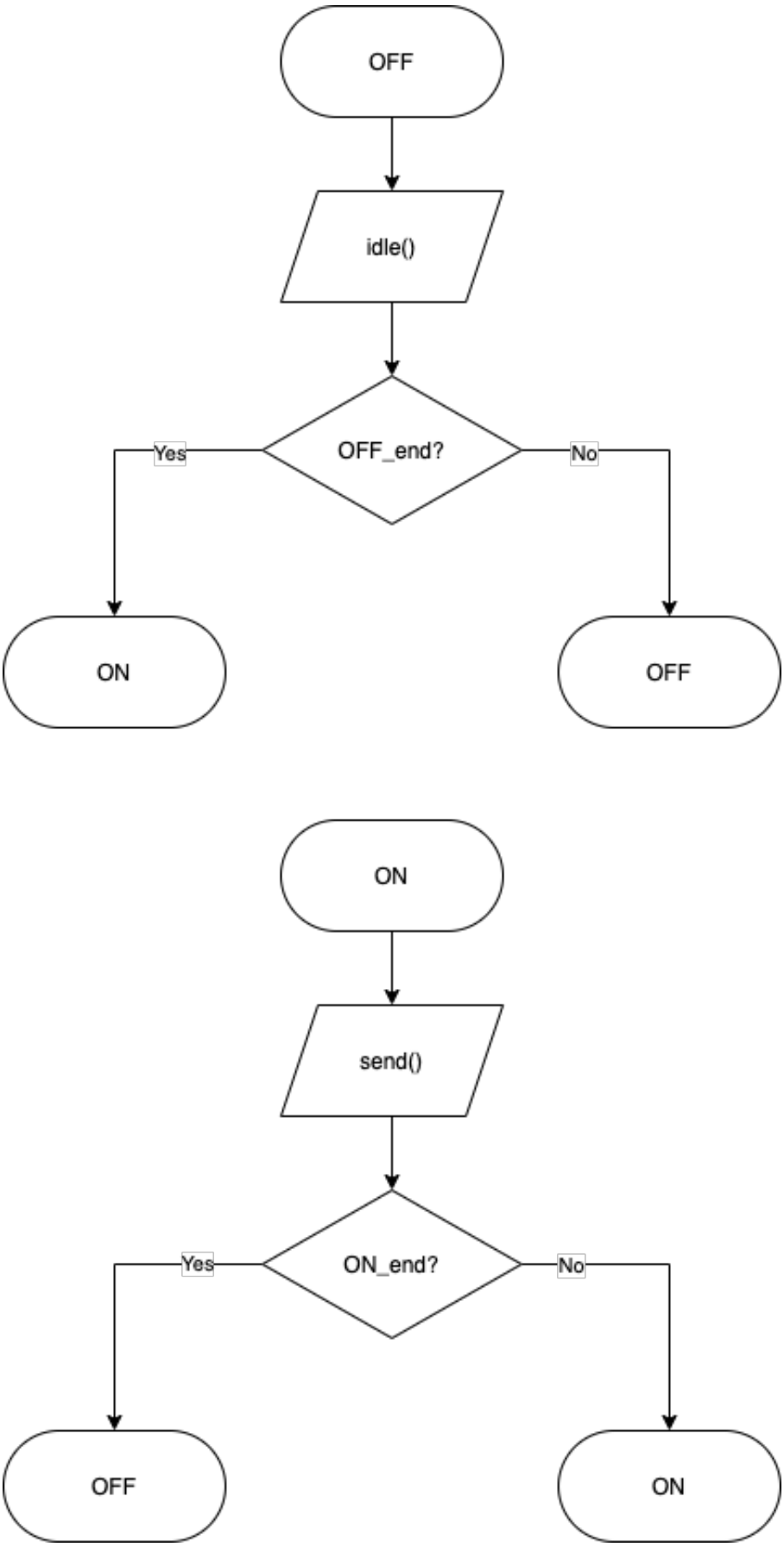
Scenariusz i algorytmy

```
napływ klientów -> kolejka -> serwer

nadawca \                        / odbiorca
nadawca -> kolejka z serwerem <-  odbiorca
nadawca /                        \ odbiorca
```

Automat przedstawiający nadawcę ON/OFF

Jeden stan będzie trwał T (np. 0.01s)



Rysunek 1: Automat przedstawiający nadawcę ON/OFF

Poglądowa implementacja takiego automatu

```
import numpy as np
import time


def idle(iterations, T):
    for i in range(iterations):
        print(f'Idle {i+1}/{iterations}')
        time.sleep(T)


def send(iterations, T):
    for i in range(iterations):
        # send data
        print(f'Sent packet {i+1}/{iterations}')
        time.sleep(T)


def main():
    T = 0.1  # state duration: 0.1s
    p_off = 0.7
    p_on = 0.3
    current_time = time.time_ns()
    end_time = current_time + 3e9  # simulation duration: 3s

    while current_time < end_time:
        off_iterations = np.random.geometric(p_off)
        idle(off_iterations, T)

        on_iterations = np.random.geometric(p_on)
        send(on_iterations, T)

        current_time = time.time_ns()


if __name__ == "__main__":
    main()
```